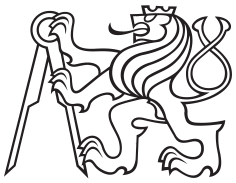
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická

Využití klávesnicového vstupu PS/2 a výstupu na LCD displej přípravku Spartan3E

Jan Skalička

Vedoucí: Ing. Pavel Lafata, Ph.D.
Květen 2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Skalička** Jméno: **Jan** Osobní číslo: **491897**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra telekomunikační techniky**

Studijní program: **Elektronika a komunikace**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Využití klávesnicového vstupu PS/2 a výstupu na LCD displej přípravku Spartan3E

Název bakalářské práce anglicky:

Using PS/2 Input and LCD Display of Spartan3E Kit

Pokyny pro vypracování:

Seznamte se s přípravkem Xilinx Spartan3E a jeho obsluhou pomocí jazyka VHDL. K přípravku připojte pomocí rozhraní PS/2 standardní klávesnici. Navrhněte a realizujte VHDL kód pro čtení stisknuté klávesy. Doplňte využití vestavěného LCD displeje s řadičem HD44780 na přípravku tak, že na tomto znakovém displeji budou jednotlivé klávesy vypisovány. Vytvořte nezbytné VHDL kódy. Výstupem tak bude jednoduché zobrazení stisknuté klávesy na LCD displeji, eventuálně jej upravte dle pokynů vedoucího práce.

Seznam doporučené literatury:

[1] Lafata, P. - Hampl, P. - Pravda, M.: Digitální technika. 1. vyd. Praha: Česká technika - nakladatelství ČVUT, 2011. 164 s. ISBN 978-80-01-04914-3.
[2] Pinker, J. - Poupa, M.: Číslicové systémy a jazyk VHDL. Praha : BEN - technická literatura, 2006. 349 s. ISBN 80-7300-198-5.
[3] Digilent: Spartan-3E Reference Manual [online]. Dostupné z: https://digilent.com/reference/programmable-logic/spartan-3e/reference-manual
[4] Popis komunikace a rozhraní PS/2 [online]. Dostupné z: https://www.avrfreaks.net/sites/default/files/PS2%20Keyboard.pdf
[5] Popis řadiče LCD HD44780 [online]. Dostupné z: https://vyvoj.hw.cz//teorie-a-praxe/dokumentace/inteligentni-displeje-a-jejich-p ripojeni-k-pc.htm

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Pavel Lafata, Ph.D. katedra telekomunikační techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **04.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

Ing. Pavel Lafata, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

.	
Datum převzetí zadání	Podpis studenta

Poděkování

Rád bych poděkoval mému vedoucímu práce Ing. Pavlu Lafatovi, Ph.D. za vedení práce, za zapůjčení veškerého potřebného hardwaru a za pomoc s řešením problémů, které se vyskytly při programování práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškerou použitou literaturu.

V Praze, 20. května 2022

Abstrakt

Práce se zaměřuje na čtení stisknuté klávesy na PS/2 klávesnici pomocí programovatelného hradlového pole FPGA. Tato přečtená klávesa je dále vypsána na LCD displej, který je součástí přípravku Spartan-3E. Displej se ovládá řadičem HD44780, který je v odvětví ovládání znakových LCD displejů jeden z nejpoužívanějších[1]. V práci je použit jazyk VHDL, v němž se inicializuje LCD displej, komunikuje s LCD displejem a komunikuje s PS/2 klávesnicí. Bylo také potřeba vyřešit vzájemné propojení jednotlivých částí VHDL kódu pro čtení dat z klávesnice a vypisování znaků na displej.

Klíčová slova: FPGA, displej, LCD displej, VHDL, HD44780, Spartan-3E, znakový displej, PS/2 klávesnice

Vedoucí: Ing. Pavel Lafata, Ph.D.

Abstract

The purpose of this project is to read a pressed key on a PS/2 keyboard using a Field Programmable Gate Array (FPGA). The pressed key is then written on an LCD display which is a part of Spartan-3E board. The display is controlled by an HD44780 controller which is one of the most used controller for controlling character LCD displays[1]. The project consists of LCD initialization, LCD communication and PS/2 keyboard communacation, all of which is written in the VHDL language. Interconnection of the parts of VHDL code for reading data from keyboard and for writing characters to the display was also needed to be solved.

Keywords: FPGA, display, LCD display, VHDL, HD44780, Spartan-3E, character diplay, PS/2 keyboard

Title translation: Using PS/2 Input and LCD Display of Spartan3E Kit

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Obvody typu FPGA se v dnešní době používají v mnoha aplikacích, mezi které patří například vytváření a testování prototypů. V této práci se využívá desky Spartan-3E, která kromě samotného čipu FPGA obsahuje i periferie, mezi které patří i dvouřádkový znakový LCD displej připojen pomocí řadiče HD44780 a PS/2 port pro připojení klávesnice nebo myši. Do tohoto portu je tedy možné zapojit klávesnici a naprogramovat FPGA tak, aby na vestavěný displej vypisovalo stisknuté klávesy.

Cílem práce je seznámit se s vývojovou deskou Spartan-3E, s jazykem VHDL a napsat VHDL kód starající se o klávesnicový PS/2 vstup, zpracování vstupních dat a vypisování znaků na LCD displej. V práci je taktéž popis problémů, které se během práce vyskytly.

Kapitola 2
Teoretický popis

2.1 FPGA

FPGA (programovatelná hradlová pole) jsou obvody, jejichž funkci je možné naprogramovat pomocí jazyka pro popis hardwaru (HDL). Kód v HDL jazyce se převede na zapojení logických hradel, které je následně implementováno do samotné desky FPGA, hradla jsou navzájem pospojována a přivedena ke vstupům a výstupům. FPGA pak vykonává námi požadovanou funkci. Oproti mikrokontrolérům či mikroprocesorům dokáže FPGA pracovat paralelně. Zatímco procesory jsou navrženy tak, že vykonávají jednu instrukci za druhou, programovatelná hradlová pole mohou v závislosti na návrhu zpracovávat vstupní signály paralelně, nezávisle na sobě. V praxi je tedy možné využívat FPGA pro urychlení určitých druhů výpočetních úloh, které by na procesoru neběžely dostatečně efektivně. Jejich programovatelnost spočívá v tom, že když je nutné, aby vykonávaly jinou činnost, dají se opakovaně jednoduše přeprogramovat, což neplatí například u zařízení typu ASIC, která programovatelná nejsou.

Typickým využitím obvodů typu FPGA je vytváření prototypů pro obvody typu ASIC. Nejprve se vytvoří návrh, který je vyzkoušen na FPGA. Pokud se při testování naleznou chyby, je možné je opravit v návrhu a opravený návrh znovu nahrát na FPGA. Teprve po odladění veškerých chyb se návrh využije pro výrobu čipů ASIC. Díky tomuto prototypování je možné ušetřit náklady, jelikož výroba vlastních ASIC čipů je nákladná a po vyrobení vadných ASIC čipů většinou nezbývá jiná možnost než vyrobit čipy zcela nové. Pro návrh moderních akcelerátorů pro umělou inteligenci nebo pro těžbu kryptoměn se tedy využívají prototypy pomocí obvodů FPGA.[3]

2.1.1 VHDL

Jazyk pro popis hardwaru (Hardware Description Language) umožňuje pomocí kódu popisovat hardwarové zapojení logických hradel v několika úrovních abstrakce. Jeden zástupce těchto jazyků je VHDL, používaný zejména v Evropě, proto se jím také v této práci zabývám. V tomto jazyce lze kódem popisovat jednotlivá logická hradla, jejich funkce a vzájemná propojení. Lze si definovat tzv. entity, které popisují konkrétní hradlo reprezentující logickou funkci.

Každá entita má v kódu definovány své vstupy a výstupy, dále pak svoji architekturu, která obsahuje implementaci vlastní logické funkce. V rámci architektury je možné využít i ostatní entity.

2.2 LCD a řadič HD44780

Displeje s technologií tekutých krystalů (LCD) se řadí k nejjednodušším možnostem zobrazení textového výstupu zařízení. V této práci se zabývám dvouřádkovým znakovým LCD displejem, kde každý řádek má místo na 16 znaků. Aby se displej dal ovládat, bývá připojen k řadiči, u těchto displejů je nejpoužívanější řadič typu HD44780[1]. S tímto typem řadiče se komunikuje pomocí 3 ovládacích signálů a 4 nebo 8 datových signálů. Zvolil jsem si komunikaci s využitím pouze 4 datových signálů, protože je takto řadič připojen k FPGA přímo v mnou používaném vývojovém kitu. Díky tomuto řadiči je možné na displej zapisovat data za pomoci několika jednoduchých příkazů s tím, že stačí použít 7 výstupních pinů.

2.2.1 Inicializace displeje

Před samotným vypisováním dat na displej je nutno displej inicializovat. Inicializace slouží k tomu, abychom displej zapnuli a nastavili mu základní vlastnosti, jako je blikání kurzoru, automatická inkrementace kurzoru, nebo posouvání obsahu celého displeje. Inicializace se provede v několika krocích, v každém kroku se řadiči pošle konkrétní bitová kombinace a po každém kroku je nutno počkat určitou dobu, kterou displej potřebuje na zpracování. Doba inicializace by se dala přibližně určit vztahem

$$t_{\text{init}} = t_{\text{start}} + 5 \cdot t_{\text{hold}} + t_1 + t_2 + t_3 + t_4 + t_{\text{clear}}. \tag{2.1}$$

Popisy veličin jsou vidět v tabulce 2.1. Před zapsáním prvních dat je ještě vhodné poslat displeji příkaz na vymazání obrazovky. Celkem se tedy pošle displeji 5 příkazů.

Veličina	Popis veličiny	Maximální doba
t_{start}	Doba zapínání displeje	15 ms
t_{hold}	Doba nutná pro správné přečtení dat řadičem	240 ns
t_1	Doba provádění prvního kroku inicializace	4,1 ms
t_2	Doba provádění druhého kroku inicializace	100 us
t_3	Doba provádění třetího kroku inicializace	40 us
t_4	Doba provádění čtvrtého kroku inicializace	40 us
t_{clear}	Doba provádění instrukce mazání displeje	1,64 ms

Tabulka 2.1: Doby trvání operací při inicializaci displeje

Obrázek 2.1: Tabulka znaků LCD displeje, převzato z [2]

2.2.2 Vypisování dat na displej

Samotné vypisování dat pak probíhá opět v několika málo krocích. Jako první je nutné vybrat adresu, kam se data budou zapisovat. Displej totiž umožňuje nadefinovat si 8 vlastních znaků, pro jejichž bitmapy má oddělenou paměť. Aby displej věděl, že na něj chceme vypisovat znaky, nikoliv si definovat vlastní, je nutné mu poslat příkaz, ve kterém definujeme adresu na displeji. Tím si také můžeme zvolit, na které místo (řádek a sloupec) bude následující znak vypsán. Po nastavení adresy už je možné odeslat displeji (resp. řadiči) kód námi požadovaného znaku k vypsání. Znaky jsou uspořádány v tabulce (viz obr. 2.1), která je podobná ASCII tabulce[2]. Po odeslání kódu znaku řadiči je znak vypsán na displej a v závislosti na předchozí konfiguraci se může inkrementovat adresa kurzoru. Díky této inkrementaci je možné poslat hned další znak, který bude zapsán na další místo na displej, aniž bychom museli explicitně nastavovat adresu kurzoru.

2.3 Klávesnice

Část projektu je zadávání dat z klávesnice. Klávesnice je jedním z nejjednodušších zařízení pro zadávání dat. Komunikace s klávesnicí je tedy poměrně jednoduchá. K připojení klávesnice k zařízení se používá několik standardů. Jedním z prvních standardů je PS/2, který se používal u počítačů dříve, než ho nahradil standard USB, který se používá doteď. Klávesnice mohou existovat i v bezdrátových podobách, těmi se ale práce zabývat nebude. V práci se používá klávesnice s připojením pomocí protokolu PS/2, jelikož se jedná o velice jednoduchý protokol se snadným zpracováním dat (například oproti protokolu USB).

2.3.1 PS/2

PS/2 je obousměrný synchronní sériový protokol. Používal se pro připojení klávesnic a myší k počítačům[4]. Dnes ho u počítačů nahradil již modernější protokol USB, nicméně s PS/2 konektorem pro klávesnici (mnohdy kombinovaným i pro myš) se můžeme stále setkat u některých základních desek. Důvodem je jeho jednoduchost a spolehlivost, oproti protokolu USB totiž nepotřebuje počítač pro komunikaci s PS/2 klávesnicí žádný ovladač[5].

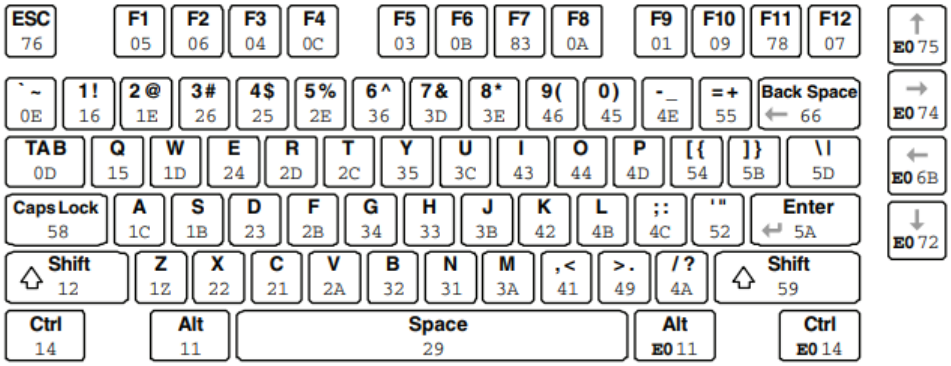
Pro komunikaci pomocí protokolu PS/2 jsou používány dva vodiče. Jeden pro přenos časových taktů (označován CLOCK či CLK), druhý pro přenos bitů dat (označován DATA). Celá sběrnice je navržena jako open-collector, v klidovém stavu je na obou vodičích tedy hodnota logické 1. Následující text se zabývá přenosem dat mezi klávesnicí a počítačem (myš není pro tuto práci důležitá).

Před tím, než začne zařízení na sběrnici vysílat, si musí zkontrolovat, zda již nevysílá druhá strana. Pokud je na vodiči CLK logická 1 po dobu alespoň 50 mikrosekund, sběrnice je volná a zařízení může začít vysílat. Při vysílání zařízení nejprve nastaví vysílaný bit na vodiči DATA, poté nastaví logickou 0 na vodiči CLOCK, to je signál pro přijímající zařízení, že může přečíst validní data z vodiče DATA, po nějaké době (maximálně desítky mikrosekund) vysílající zařízení nastaví na vodiči CLOCK opět logickou 1 a připraví se na vysílání dalšího bitu[4].

Jak již bylo zmíněno, protokol PS/2 je protokol obousměrný. Opačný směr dat - z počítače ke klávesnici - je také používaný, konkréně pro nastavení stavových LED diod na klávesnici (typicky indikace Num Lock, Caps Lock, Scroll Lock). Proto je také ve sběrnici PS/2 použita metoda open-collector. Tato práce se ale komunikaci směrem ke klávesnici dále nezabývá.

Struktura odesílaných dat

Komunikace klávesnice po protokolu PS/2 probíhá v 11bitových rámcích. Po stisknutí tlačítka na klávesnici tedy klávesnice odešle 11 bitů po PS/2 sběrnici. První bit je startovací bit, je vždy 0. Po něm následuje 8 datových bitů, první se odesílá LSB (nejméně významný bit). Další bit je paritní bit. Na konec



Obrázek 2.2: Scankódy kláves na klávesnici, převzato z [2]

se odešle ukončovací bit, ten je vždy 1. Každá klávesa má svůj unikátní kód. Některé klávesy (zejména speciální klávesy) nemají pouze 8bitový kód, ale kód o délce násobku 8 bitů. Tyto kódy jsou pak posílány po sobě, vždy po 11 bitech (tedy včetně startovacího, paritního a ukončovacího bitu)[4]. O kódech kláves (tzv. scankódech) píšu v následující sekci. Startovací a ukončovací bity samotné nenesou žádnou informaci, jsou redundantní, mohou ale posloužit k synchronizaci a k detekci chyb při přenosu. Paritní bit je typu lichá parita, to znamená, že počet jedniček v osmi bitech kódu klávesy a paritním bitu musí být liché číslo. Pokud by se při přečtení zjistilo, že parita nesouhlasí, je zřejmé, že došlo k chybě při přenosu dat, a bylo by vhodné tato data ignorovat. Zároveň je ale 1 paritní bit příliš málo informací na to, aby se dalo poznat, kde k chybě došlo. Není tedy možné správná data zrekonstruovat.

Pro případ přečtení chybného snakódu je v PS/2 implementován příkaz Resend[2]. Jedná se o příkaz (hexadecimálně zapsaný `FE`), který může odeslat počítač klávesnici a klávesnice na něj odpoví zopakováním posledního odeslaného scankódu. Tento příkaz ale v mé práci pro jednoduchost neimplementuji.

2.3.2 Scankódy kláves

V předchozím textu bylo nastíněno, že každá klávesa má svůj unikátní kód. Anglicky se nazývá scancode, do češtiny ho lze přeložit jako snímací kód či scankód. Na základě tohoto kódu je v počítači vyhodnoceno, jaká klávesa byla stisknuta. Scankódy kláves byly navrženy tak, aby bylo co nejjednodušší je implemetovat fyzicky na klávesnici. Na obrázku 2.2 je vidět, že klávesy, které jsou fyzicky blízko (pod sebou), mívají podobné scankódy (např. scankódy kláves Q, W, A, S a Z všechny začínají jedničkou - mají tedy společné 4 bity). Některé speciální klávesy (např. šipky) mají scankód složený ze dvou 8bitových skupin. Pro šipky je první skupina hexadecimálně zapsaná jako `E0`, poté až je unikátní kód každé šipky. Systém scankódů je navržen tak, že každý kód je unikátní a nezaměnitelný s kódem jiné klávesy. Například tedy nenajdeme klávesu, která by měla kód samotné `E0`, jelikož by pak nebylo jasné, zda byla zmáčknuta tato klávesa, nebo třeba šipka.

2.4 Spojení klávesnice a displeje

Cílem práce je propojit klávesnici s diplejem tak, že po stisknutí tlačítka na klávesnici se na displeji vypíše stisknutý znak. Bude tedy možné na displej psát pomocí připojené PS/2 klávesnice. FPGA tedy bude hrát roli počítače, v čemž není problém, právě díky tomu, že je protokol PS/2 jednoduchý. Vše, co bylo v předchozích kapitolách o klávesnici napsáno o počítači, je možné vztáhnout i na FPGA. V FPGA je tedy snadné protokol PS/2 implementovat, v této práci stačí pouze část čtení dat. Odesílat data klávesnici pro rozsvícení indikačních LED diod není důležité a nebude to tedy využito. Kromě čtení dat se data v FPGA také budou zpracovávat. V rámci zpracování je taktéž možné detekovat speciální klávesy a na základě nich měnit chování programu.

2.4.1 Rozdělení VHDL kódu

Zdrojový VHDL kód celého projektu je rozdělen do dvou základních částí. Jedna část ovládá LCD displej, stará se tedy o inicializaci a vypisování textu na displej. Tato část bude pouze dostávat informace o tom, jaké znaky má na displej vypsat, samotné zpracování dat v ní nebude.

Druhá základní část se stará o komunikaci s PS/2 klávesnicí, respektive o čtení dat z klávesnice. Po přijetí dat je nutné zkontrolovat validitu dat (tj. startovací bit, ukončovací bit a potenciálně paritu). Pokud budou přijatá data validní, projdou zpracováním, to znamená přeložením scankódu na znak vypsatelný na displej, a následně budou odeslána přes signál do části ovládající displej.

Pro podporu funkce a zlepšení čitelnosti VHDL kódu je možné využít funkcí v jazyce VHDL, konkrétně například pro překlad scankódu na kód pro displej. Funkce je samostatná část kódu, která na základě vstupu vrátí výstup. Definice chování funkce se nenachází přímo uvnitř procesu, to vede k lepší čitelnosti kódu procesu, protože v něm je obsaženo pouze volání dříve definované funkce.

Kapitola 3
Praktická část

3.1 Vypisování znaků na LCD

Pro začátek jsem se rozhodl implementovat komunikaci s LCD displejem. V dokumentaci k vývojové desce[2] je popsáno, jak přesně má vypadat inicializace displeje. Jedná se o poslání několika příkazů po sobě, jak je již popsáno v teoretické části. Pro zasílání inicializačních příkazů jsem si vytvořil jednoduchý stavový automat. Každý ze stavů má za úkol odeslat displeji pevně daný příkaz nebo čekat pevnou dobu, kterou displej potřebuje k vykonání předešlého příkazu. Mezi jednotlivými stavy se přechází automaticky po uběhnutí času, který má každý stav nastaven zvlášť. Například stav `wait_15ms` má za úkol počkat 15 ms na to, až displej po připojení napájení přejde do stavu, kdy je schopen přijímat příkazy. Tento stav tedy čeká, až proběhne 750000 cyklů, což při 50MHz frekvenci hodinového signálu odpovídá 15 ms. Po dosažení cyklu číslo 750000 se čítač cyklů vynuluje a přejde se na následující stav. V těchto krocích se displej připraví již na příkazy na konfiguraci či samotné vypisování znaků.

Po těchto několika krocích se musí displeji nastavit základní konfigurace. Displej se tedy například musí zapnout, musí se mu sdělit, že se s ním bude komunikovat po 4 vodičích namísto 8 a dají se nastavit další detaily jako je blikání kurzoru nebo automatická inkrementace pozice kurzoru. Po tomto nastavení je tedy displej úspěšně připraven na příjem příkazů k vypisování znaků. Pro implementaci odesílání těchto konfiguračních příkazů jsem si vytvořil druhý stavový automat, který se opět mezi stavy přesouvá automaticky s plynoucím časem. Po konfiguraci displeje se skončí ve stavu `idle_2`, ve kterém se čeká, až přijdou data z další části kódu - tedy z klávesnice. Pro testovací účely jsem ale nejprve na displej vypisoval testovací zprávu. Tím jsem si ověřil, že můj kód pro inicializaci displeje a zapisování znaků na displej funguje validně.

3.2 Čtení znaků z PS/2 klávesnice

Data z klávesnice přichází ve dvou vodičích. Jeden určuje hodinový takt, druhý přenáší samotná data o stisknuté klávese. Jelikož se zabývám pouze

jednosměrným přenosem dat směrem z klávesnice k FPGA, mohu obě datové linky považovat za vstupní, bude je tedy řídit vždy pouze klávesnice. Z popisu PS/2 protokolu[2] vyplývá, že klávesnice data na datovou linku zapisuje tehdy, když je na časové lince hodnota logické jedničky, poté teprve nastaví na časové lince logickou nulu. Mezi sestupnou hranou časového signálu a další náběžnou hranou časového signálu je okno 5 až 25 mikrosekund, kdy jsou data validní, a jsou tedy určena k přečtení počítačem nebo v mém případě deskou FPGA. Rozhodl jsem se tedy detekovat sestupnou hranu a při ní data z datové linky vyčíst. Tato data nadále ukládám do 11bitového bufferu a po přečtení 11. bitu s nimi dále pracuji.

Detekci sestupné hrany lze vytvořit jednoduše. Frekvence vnitřního krystalu vývojové desky je 50 MHz. Maximální frekvence, kterou komunikuje PS/2 protokol je nižší než 20 kHz. Není tedy vůbec problém PS/2 časovou sběrnici vzorkovat při každé náběžné hraně 50MHz vestavěného časového signálu. Pro tyto účely jsem si vytvořil proces, který má v citlivostním seznamu proměnnou `Clock`. Tento proces se tedy spouští vždy při změně logické hodnoty proměnné `Clock`, která je napojena na vnitřní 50MHz oscilátor. Uvnitř tohoto procesu jsem si definoval proměnnou `PS2_CLK_last`, jejímž úkolem je uchovávat logickou hodnotu časové sběrnice PS/2 z posledního průběhu cyklu. Tuto proměnnou tedy na konci procesu nastavím na aktuální hodnotu časového signálu PS/2 a tím docílím toho, že při dalším průběhu tohoto procesu v ní bude k dispozici signál starý 20 nanosekund (při 50MHz oscilátoru odpovídá perioda mezi dvěma náběžnými hranami 20 ns). To je zásadně důležité k možnosti detekce sestupné hrany PS/2 časového signálu. V procesu totiž porovnám aktuální hodnotu přečtenou z PS/2 časového signálu s hodnotou z minulého cyklu a pakliže byla stará hodnota logická 1 a aktuální hodnota logická 0, proběhla v posledních 20 nanosekundách sestupná hrana, kterou potřebuji detekovat. Zjednodušený kód ve VHDL vypadá takto:

```
process(Clock)
    variable PS2_CLK_last : std_logic := '1';
begin
    if PS2_CLK = '0' and PS2_CLK_last = '1' then
        -- detekovana sestupna hrana na PS/2 casovem signalu
    end if;
    PS2_CLK_last := PS2_CLK;
end process;
```

Pro začátek jsem si data vypisoval na displej v binární formě. Tím jsem si ověřil, že i tato část kódu mi fungovala. Všechny 11bitové skupiny musí splňovat známé podmínky. První bit (start bit) musí být vždy 0, poslední bit (stop bit) musí být vždy naopak 1. Vypisování dat z klávesnice na displej se sice dařilo, občas se ale objevila chyba, nebyly splněny podmínky popsané v předchozí větě. Tento problém bylo nutné vyřešit, jeho řešení popisuji v následující sekci.

3.3 Problémy se čtením dat

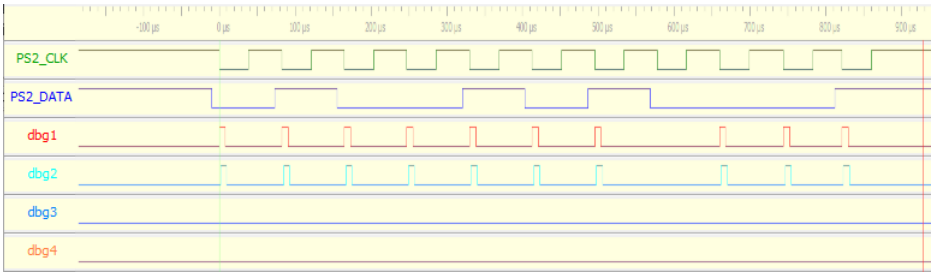
Při stisknutí tlačítka na klávesnici se stávalo, že se nepřečetla validní data. To mělo za následek špatné vyhodonocení scankódu a tím i nemožnost vypsat na displej správný znak. Z pozorování to vypadalo, že některé bity, které měly přijít z klávesnice, nepřišly, resp. nebyly detekovány a uloženy do paměti. To, který bit se vynechal, bylo náhodné. Někdy se stalo, že bit vypadl jeden, někdy dva a někdy dokonce tři. Někdy ale nevypadl žádný a data byla přečtena celá a správná. Výpadek byť jednoho jediného bitu znamená nemožnost správné interpretace dat, tudíž bylo zásadní tuto chybu odladit a opravit.

Zkoušel jsem svůj kód ladit mnoha způsoby. Nejprve jsem si vypisováním všech 11 přečtených bitů na displej zjistil, co se skutečně přečte. Z toho jsem usoudil, že se někdy vynechá jeden nebo více bitů, jak jsem psal v minulém odstavci. Někdy se tak stávalo při 1 z 10 stisků klávesy, jindy byly ale chyby mnohem častější, třeba v 1 ze 3 případů. Jako první mě napadlo, že by se na PS/2 sběrnici mohly objevovat nějaké zákmity, kvůli kterým by se data nečetla správně. Abych tuto možnost eliminoval, implementoval jsem do VHDL kódu podmínku, že se data přečtou až tehdy, když je po sestupné hraně na časové sběrnici stálá logická 0 po dobu 2 mikrosekund. Pokud je i po 2 mikrosekundách časový signál stále na hodnotě logické 0, pak mohu data přečíst. Tato změna bohužel nepřinesla žádný pokrok. Jako další jsem zkusil počítat počet detekovaných sestupných hran a aktuální součet vypisovat na LCD displej. Tato změna mi velmi pomohla, protože jsem díky ní viděl, že když se přečtou špatná data, není počet detekovaných sestupných hran 11. Buď jich bylo detekováno méně, v tom případě se na displej nevypsala data žádná, protože se stále čekalo na 11. přečtený bit, nebo jich bylo detekováno více, kdy se detekovaly sestupné hrany z dalšího scankódu odeslaného hned za prvním z důvodu delšího stisknutí klávesy. Z tohoto ale vyplynulo, že problém bude někde v detekci sestupné hrany.

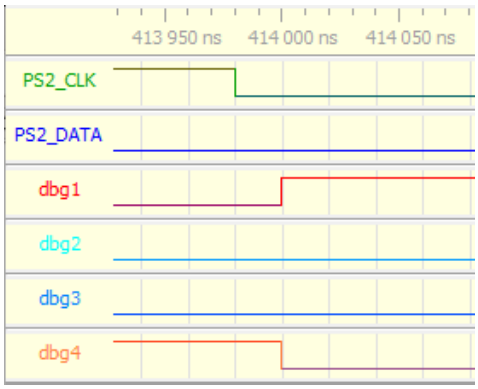
3.3.1 Logický analyzátor

Měl jsem nápad, že bych celý kód pro čtení PS/2 dat napsal ve VHDL znovu, snad bez chyb, ale nakonec jsem zvolil jinou cestu. Na návrh mého vedoucího práce jsem si ze školy vypůjčil logický analyzátor SIGMA od firmy ASIX, se kterým jsem začal znovu ladit celý program. Vyvedl jsem si několik testovacích signálů, které jsem použil pro detekci, zda se kód dostal do určitého místa. Například jeden signál jsem využil pro signalizaci detekce sestupné hrany, druhý signál jsem využil pro signalizaci toho, že po sestupné hraně zůstal PS/2 časový signál 2 mikrosekundy stále na úrovni logické 0, třetí signál jsem využil pro signalizaci úspěšného přečtení všech 11 bitů a čtvrtý signál jsem využil jako signalizaci úspěšného odeslání dat displeji. Kromě toho jsem si zapojil přímo i signály PS/2 - tedy časový a datový. V softwaru SIGMA Logic Analyzer jsem si pak všechny tyto signály nechal vykreslit.

Mezi užitečné výhody logického analyzátoru SIGMA patří[6] paměť na vzorky s kompresí, díky které se do paměti vejde velké množství vzorků. Je



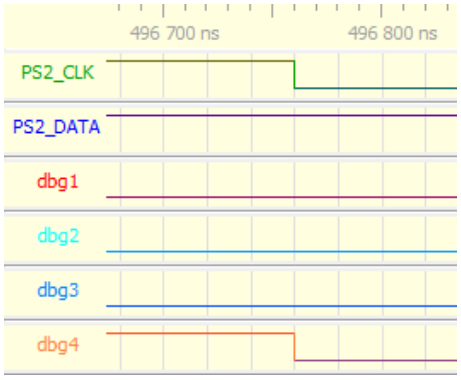
Obrázek 3.1: Jedna sestupná hrana PS/2 časového signálu nebyla detekována



Obrázek 3.2: Zpoždení PS2_CLK_last o 20 ns

také možné zobrazit průběh i před spouštěcí (trigger) podmínkou tak, že počet vzorků před triggerem bude stejný jako počet vzorků po triggeru, díky tomu je snadné vidět všechny signály před samotným spuštěním. Jako spouštěcí podmínku jsem si nastavil sestupnou hranu hodinového signálu PS/2 a nechal si vykreslit několikrát vždy signály po jednom stisknutí tlačítka na klávesnici. Výsledek jednoho z chybných čtení vidíme na obrázku 3.1. Vidíme, že bylo detekovaných 6 sestupných hran, pak jedna ne a pak další 4. V tomto je tedy ten problém. V kódu pro detekci sestupné hrany je tedy nějaký problém.

Poupravil jsem signály pro ladění tak, že na čtvrtém ladicím signálu `dbg4` bude vždy aktuální hodnota proměnné `PS2_CLK_last`. Správně bych tedy v programu měl vidět časový signál PS/2 a ladicí signál, který je o 20 ns opožděný, jako tomu je na obrázku 3.2. Na tomto obrázku je také vidět, že na signálu `dbg1` je signalizována detekce sestupné hrany. V případech, kdy se sestupná hrana nedetekovala se ale ukázalo, že proměnná `PS2_CLK_last` v sobě nemá signál o 20 ns opožděný, ale signál stejný, jako je aktuálně na časové sběrnici PS/2, to je vidět na obrázku 3.3. Pokud nastane tato situace, není možno podmínkou kontrolovat, jaká byla předchozí hodnota časové sběrnice PS/2, protože informaci o předchozí hodnotě už nemám. Z tohoho důvodu pak není možné sestupnou hranu detekovat, čímž nepřečtu jeden bit, bez kterého ale nepřečtu správná data a nemohu je tedy vypsat na displej.



Obrázek 3.3: Signál PS2_CLK_last není zpožděn

3.3.2 Řešení problému

Chyba je tedy v kódu, který má za úkol detekovat sestupnou hranu časového signálu PS/2. Není zaručeno, že nastavování hodnoty proměnné `PS2_CLK_last` na aktuální hodnotu časového signálu PS/2 proběhne až po tom, co se vyhodnotí podmínka `if PS2_CLK = '0' and PS2_CLK_last = '1' then`. Bylo tedy třeba vymyslet řešení detekce sestupné hrany jinak. Mě napadla myšlenka, že ačkoliv není zaručeno pořadí nastavování hodnoty proměnné a vyhodnocování podmínky, je zaručeno, že se proměnná nastaví v daném cyklu právě jednou. Napadlo mě tedy, že kdybych si vytvořil posuvný registr, mohl bych v něm uchovávat delší historii časového signálu PS/2 než pouze o jeden cyklus. Výhoda tohoto řešení je ta, že když bych měl posuvný registr o délce 2, měl bych k dispozici hodnotu, která byla na časovém signálu PS/2 před 40 ns nebo 20 ns, kdyby se posuvný registr aktualizoval dříve než by se vyhodnocovala podmínka. Nemohla by ale nastat situace, že bych neměl k dispozici zpožděný signál.

Implementoval jsem tedy posuvný registr, zvolil jsem délku 6, ačkoliv by měla stačit klidně i délka 2. O trochu delší registr ničemu neuškodí. Zjednodušený kód detekce sestupné hrany upravený pro správnou funkčnost pomocí posuvného registru je možno vidět zde:

```
process(Clock)
    variable PS2_CLK_history : std_logic_vector(5 downto 0);
begin
    if PS2_CLK = '0' and PS2_CLK_history(5) = '1' then
        -- detekovana sestupna hrana na PS/2 casovem signalu
    end if;
    PS2_CLK_history := PS2_CLK_history(4 downto 0) & PS2_CLK;
end process;
```

Po implementaci jsem celý kód vyzkoušel a problém zmizel.

3.4 Překlad scankódu na znak k vypsání

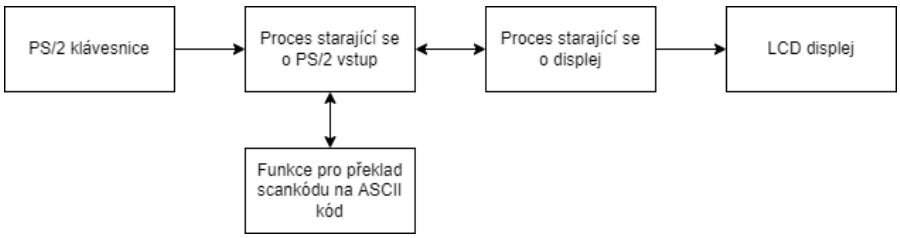
Výstupem procesu, který čte data z PS/2 sběrnice je 11 bitů. První a poslední bit jsou pevně dány, pokud jsou správné, nenesou žádnou informaci a dále se jimi zabývat nemusíme. Paritní bit je vhodný nástroj pro odhalení chyby čtení dat. Zatím jsem ale kontrolu parity neimplementoval, bit tedy prozatím ignoruji. Zbývá tedy 8 datových bitů, které označují samotný scankód stisknuté klávesy. LCD displej ovšem těmto scankódům nerozumí, aby bylo tedy možné vypisovat stisknuté klávesy na displej, je nutno provádět překlad scankódu na kód, kterému LCD displej rozumí. Vytvořil jsem si tedy funkci, která z 8 bitů scankódu (viz obr. 2.2) udělá 8 bitů kódu podobného ASCII kódu (viz obr. 2.1), které když pošlu na displej, zobrazí se požadovaný znak. Písmeno Q má například scankód hexadecimálně `15`, to je binárně `00010101` a odpovídající kód pro displej je binárně `01010001`. Obdobná pravidla lze sestavit pro všechna písmena. Překladová funkce vypadá zjednodušeně tedy takto:

```
function ScancodeToAscii(Scancode : std_logic_vector(7 downto 0))
  return std_logic_vector is
begin
    case Scancode is
        when "00010101" => return "01010001"; -- Q
        -- ...
        when "00111010" => return "01001101"; -- M
        when "00101001" => return "00100000"; -- Space
        when others => return "00111111"; -- '?' neznamy scankod
    end case;
end function;
```

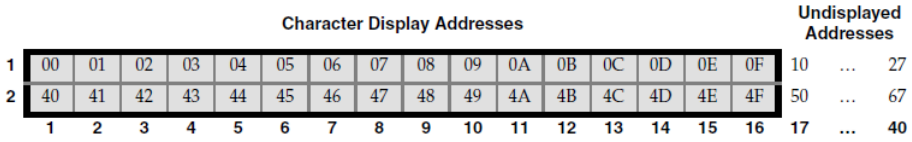
3.5 Komunikace mezi procesy

Kód mám rozdělen na dva nezávislé procesy. Jeden se stará o vypisování dat na displej včetně prvotní inicializace a následného odesílání řídicích příkazů. Druhý zpracovává příchozí data z PS/2 klávesnice. V jazyce VHDL probíhá komunikace mezi procesy pomocí tzv. signálů. Signál je entita podobná proměnné, avšak narozdíl od proměnné lze signál využít ve více procesech najednou[7]. Zapisovat do signálu lze ovšem pouze z jednoho procesu, ostatní mohou data pouze číst. Signál je reprezentace vodiče v jazyce VHDL. Pro účely této práce jsem si navrhnul několik signálů pro přenos dat mezi oběma procesy. Osmibitový signál `write_ascii_character_signal` slouží k odesílání dat o znacích, které má displej vypsat. Do tohoto signálu se zapisuje 8 bitů výstupu funkce `ScancodeToAscii`, která je volána po úspěšném přečtení scankódu z PS/2 sběrnice.

Další signál je jednobitový `keyboard_ready`, do kterého je zapsána logická 1 po zapsání dat do signálu `write_ascii_character_signal`. Slouží tedy jako signalizace toho, že byla z procesu starajícího se o čtení dat z klávesnice odeslána data o přečteném znaku. Tento signál sleduje proces starající



Obrázek 3.4: Struktura řešení, nakresleno pomocí nástroje draw.io



Obrázek 3.5: Hexadecimální adresy znaků displeje, převzato z [2]

se o displej. Jakmile na tomto signálu detekuje logickou 1, začne na LCD displej odesílat příkazy pro vypsání správného znaku, který získá ze signálu `write_ascii_character_signal`. Po vypsání znaku na displej se na hodnotu logické 1 nastaví třetí signál `keyboard_clear`. Tento signál slouží ke komunikaci opačným směrem. Jeho hodnotu tedy nastavuje proces starající se o displej. Naopak proces starající se o klávesnici tento signál sleduje a při detekci logické 1 na tomto signálu zapíše hodnotu logické 0 do signálu `keyboard_ready`. To je nutné, aby se na displej nevypisoval do nekonečna stejný znak.

Proces starající se o displej nakonec detekuje, že se objeví na signálu `keyboard_ready` opět hodnota logické 0 a signálu `keyboard_clear` nastaví taktéž hodnotu na logickou 0. Tímto je vypsání znaku dokončeno a program je připraven na stisk další klávesy. Struktura řešení a propojení částí kódu je vidět na obrázku 3.4.

3.6 Adresace displeje

Displej na vývojovém kitu má 2 řádky po 16 znacích. Každý znak má svoji adresu, na kterou lze přesunout kurzor pomocí příkazu odeslanému displeji. Po vypsání jednoho znaku na displej se při správném nastavení adresa inkrementuje. Další znak bude tedy vypsán na další pozici. Po zapsání všech 16 znaků v prvním řádku se ale nenastaví adresa na adresu prvního znaku druhého řádku. Jak je vidět na obrázku 3.5, za každým řádkem se nachází ještě několik adres, které nejsou zobrazovány přímo na displeji. Tyto adresy by se zobrazovat mohly, kdyby se v konfiguraci displeje zvolilo posouvání obsahu při zápisu znaků. Toto nastavení ale v mé práci nepoužívám. Po zapsání 16 znaků na řádku se tedy dalších 24 znaků zapisuje mimo zobrazovanou plochu displeje, to je potřeba opravit.

Řešení, které jsem zvolil, je jednoduché. Vytvořil jsem si proměnnou `written_chars`, kterou inkrementuji pokaždé, kdy se na displej vypíše znak.

Když se zapíše 16 znaků, odešle se displeji příkaz pro přesun adresy na adresu prvního znaku druhého řádku. Další znaky se tedy vypisují na druhý řádek. Když se zapíše 32 znaků, dostali bychom se opět za konec řádku, tentokrát druhého, v tento moment se tedy displeji nastaví adresa opět na začátek prvního řádku a zároveň se vynuluje proměnná `written_chars`. Díky tomuto řešení se tedy znaky nikdy nevypisují mimo zobrazovanou plochu a psaní na displej je tak mnohem pohodlnější. Nastavení adresy displeje (tedy kurzoru) se provádí ve stavech `set_dd_ram_line_1` a `set_dd_ram_line_2`. V názvu vyskytující se `set_dd_ram` odkazuje na název instrukce `Set DD RAM Address`, která nastavuje hodnotu registru adresy Display Data RAM, což je paměť, ve které jsou uloženy znaky vypsané na displej. Ukázka kódu je vidět zde:

```
write_state := idle_2; -- bezne chovani, idle_2 je stav
                       -- cekani na novy znak
-- inkrementovat written_chars
written_chars := written_chars + 1;
if written_chars = 16 then -- presunout na radek 2
  write_state := set_dd_ram_line_2;
elsif written_chars = 32 then -- presunout na radek 1
  write_state := set_dd_ram_line_1;
  written_chars := 0;
end if;
```

3.7 Podpora klávesy Enter

Jedno vylepšení, které mě napadlo, že by bylo možné pro zpříjemnění psaní implementovat, je podpora klávesy Enter. Tato klávesa by mohla po stisknutí přesunout kurzor na začátek druhého řádku, když se kurzor nachází kdekoliv v prvním řádku, nebo na začátek prvního řádku, když se kurzor nachází kdekoliv v druhém řádku. Díky vylepšení z minulé kapitoly je implementace tohoto vylepšení velmi jednoduchá. O skok na další řádek se musí starat proces, který ovládá celý LCD displej. Detekce stisknutí klávesy Enter ovšem lze provádět v procesu starajícím se o PS/2 vstup. Byla by tedy možnost vytvořit si nový signál, do kterého by proces starající se o PS/2 vstup zapisoval příznak, že byl Enter stisknut. Tento signál by pak sledoval proces starající se o displej a na základě tohoto příznaku by proces poslal na displej odpovídající příkaz pro změnu adresy. Toto řešení mi ale přišlo zbytečně složité, protože bych musel podobně jako u signálu `keyboard_ready` řešit i jeho resetování, tudíž by musel vzniknout další signál analogický k signálu `keyboard_clear`, který by musel zase sledovat proces starající se o PS/2 vstup.

Existuje ale jednodušší řešení, které by využilo všech stávajících signálů a nevyžadovalo signály nové. Do funkce `ScancodeToAscii` jsem přidal nový řádek pro klávesu Enter. Není na něm nic speciálního, pouze se scankód klávesy Enter (hexadecimálně `5A`) přeloží na ASCII kód znaku new line (hexadecimálně `0A`). V procesu starajícího se o PS/2 vstup se tedy nic nezmění.

```
when "01011010" => return "00001010"; -- Enter
```

Znak pro nový řádek se tedy odešle procesu starajícímu se o displej jako každý jiný znak. Tento proces ale znak pro nový řádek detekuje jako speciální znak a místo jeho vypsání na displej displeji odešle příkaz pro změnu řádku. Kód pro tuto detekci vypadá zjednodušeně takto:

```
if write_ascii_character_signal = "00001010" then
  -- 0A hexadecimalne odpovida ASCII kodu new line znaku
  write_state := switch_lines;
else
  write_state := write_data;
end if;
```

Stav `write_data` vypíše na displej normální znak. Pokud se ale v signálu `write_ascii_character_signal` vyskytuje ASCII kód `00001010` (odpovídá znaku new line – nový řádek), je změněn stav stavového automatu na `switch_lines`. Tento stav slouží pro zjištění, na kterém řádku se aktuálně nachází kurzor a pro nastavení správné adresy. Kód je vidět v následujícím úryvku:

```
when switch_lines =>
  -- zmenit radky podle written_chars
  if count = 0 then
    keyboard_clear <= '1'; -- vymazat priznak keyboard_ready
  elsif count = 8 then
    count := -1;
    if written_chars >= 0 and written_chars <= 15 then
      -- presunout na radek 2
      write_state := set_dd_ram_line_2;
      written_chars := 16; -- nastavit spravnou hodnotu
    else
     -- presunout na radek 1
      write_state := set_dd_ram_line_1;
      written_chars := 0; -- nastavit spravnou hodnotu
    end if;
  end if;

  count := count + 1;
```

Pomocí klávesy Enter je tedy nyní možné přesunout se na nový řádek bez nutnosti aktuální řádek dopisovat.

3.8 Problém s pomalým psaním

V nynějším stavu tedy bylo možné na klávesnici pohodlně psát. Muselo se ale psát poměrně pomalu. Když se psalo rychle, některá část kódu nepracovala správně a na displej se vypisovaly otazníky (to odpovídá přečtení neznámého scankódu z klávesnice). Dosud jsem měl rychlost psaní v kódu limitovanou

pouze proměnnou `cooldown`, která se starala o to, že po přečtení validního kódu se dalších 100 ms dat na PS/2 sběrnici bude ignorovat. To jsem implementoval z důvodu, aby se nečetly duplicitní scankódy, když budu jednu klávesu držet stisknutou déle (nepovede se mi ji stisknout a pustit dostatečně rychle). Toto řešení se ukázalo jako nedokonalé.

3.8.1 Chování PS/2 při rychlém psaní

Jak se lze dočíst z dokumentace k vývojové desce[2], při držení klávesy PS/2 klávesnice posílá opakovaně scankód držené klávesy každých přibližně 100 ms. Po puštění dané klávesy klávesnice pošle kód `F0` následovaný scankódem puštěné klávesy. Já jsem potřeboval zjistit, co přesně se děje při rychlém psaní na klávesnici, proto jsem znovu použil logický analyzátor, ve kterém jsem zjistil následující chování. Pokud se stiskne klávesa a po ní rychle (přibližně 100 ms) klávesa další, klávesnice nejdříve pošle kód `F0`, pak scankód puštěné klávesy a po nějaké chvíli (přibližně 20 ms) teprve scankód druhé stisknuté klávesy. Toto v mém kódu vyvolá chybu, protože jednak scankód `F0` není známý kód klávesy, na displej se tedy vypíše otazník, a jednak scankód druhé stisknuté klávesy je ignorován, protože je odeslán dříve než uplyne 100 ms ignorování PS/2 vstupu (určené proměnnou `cooldown` popisovanou v předchozím odstavci). Na displej se tedy ve výsledku místo některých správných znaků vypisují otazníky.

Pokud se na klávesnici píše pomaleji (maximálně 4 údery za sekundu), tento problém se nevyskytuje. Bylo by tedy možné smířit se s touto mírnou nedokonalostí, já jsem se ale pokusil vymyslet jednoduché řešení tohoto problému.

3.8.2 Řešení rychlosti psaní

Na základě sledování průběhu PS/2 signálů v logickém analyzátoru jsem vytvořil návrh, jak upravit jednotlivé časové parametry, abych dokázal detekovat všechny scankódy stisknutých kláves a zároveň ignorovat ty, které nepotřebuji. Jako první jsem změnil hodnotu proměnné `cooldown` tak, aby se nastavovala na pouhých 50 ms ignorování PS/2 vstupu. Díky tomuto mi ani při rychlém psaní neuteče kód `F0`, který potřebuji detekovat. Při detekci kódu `F0` jsem v kódu udělal tu změnu, že tento konkrétní kód se nepokusím vypisovat na displej, ale budu ho ignorovat. Současně ale nastavím proměnnou `cooldown` tak, aby byla data na sběrnici PS/2 ignorována 10 ms, tím docílím toho, že nebudu číst scankód puštěné klávesy. Naopak ale přečtu scankód další stisknuté klávesy, který se na PS/2 sběrnici objeví po přibližně 20 ms po kódu `F0`. Část VHDL kódu pro tuto detekci je vidět zde:

```
if ps2_data_buffer = "11111100000" then  -- precten kod F0
  cooldown := 500000; -- cooldown 10 ms a ignorovat
else -- precten scankod klavesy
  -- ...
end if;
```

Kód `F0` je v podmínce zapsán obráceně, než se objevuje na PS/2 sběrnici. To je určeno deklarací proměnné `ps2_data_buffer` jako `std_logic_vector(10 downto 0)`. Po této úpravě je možné psát na klávesnici velmi rychle, nestalo se mi, že by mě rychlost kódu nějak omezovala a já kvůli ní musel zpomalovat mačkání kláves. Psaní je tedy opět o něco pohodlnější.

3.9 Podpora klávesy Shift

Dosud se na displej dala vypisovat pouze velká písmena. Napadlo mě ale vylepšení, které by umožnilo psaní velkých i malých písmen. Můj návrh byl takový, že by se detekovalo stisknutí klávesy Shift (resp. kterékoliv ze dvou kláves Shift na klávesnici) a následující stisknuté písmeno by se vypsalo velké. Pro implementaci tohoto vylepšení jsem si vytvořil novou proměnnou `shift_button`, která se při detekci stisknutí Shiftu nastaví na hodnotu logické 1 (případně na hodnotu logické 0, jestliže hodnotu logické 1 měla). Při stisknutí následujícího písmena se pak změní příslušný ASCII kód v závislosti na tom, zda je hodnota proměnné `shift_button` logická 1.

3.9.1 Návrh ASCII tabulky

LCD displej s řadičem HD44780 používá jako tabulku znaků upravenou ASCII tabulku (viz obrázek 2.1). Tato tabulka byla navržena tak, aby rozdíl mezi velkými a malými písmeny byl právě v jednom bitu. Binární reprezentace velkého písmena A je `01000001`, zatímco malého písmena a je `01100001`. Všechna malá písmena mají hodnotu třetího bitu zleva logické 1, všechna velká písmena mají naopak třetí bit zleva roven 0. Tento návrh je velice vhodný, protože díky němu lze jednoduchou operací přecházet mezi malými a velkými písmeny.

3.9.2 Úprava funkce pro překlad

Nejvhodnější místo pro implementaci velkých a malých písmen je v mém kódu samotná funkce `ScancodeToAscii`, která se stará o vytváření ASCII kódů pro vypsání na displej. Na vstup této funkce jsem tedy kromě scankódu přidal další proměnnou s názvem `shift_character`. Tato proměnná je funkci předávána a je v ní k dispozici jeden bit reprezentující to, zda má být vypsáno velké písmeno či malé písmeno. Dále je uvnitř funkce definována proměnná `shift_mask` o šířce 8 bitů, která obsahuje buď 8 nul, pokud je v proměnné `shift_character` logická 1, nebo jedničku na třetím bitu zleva v případě opačném. Každý řádek obsahující scankód a odpovídající ASCII kód pak tuto masku přidává k vracenému ASCII kódu logickou operací OR. Pro představu jednotlivé řádky vypadají takto:

```
when "00010101" => return "01010001" or shift_mask; -- Q
when "00011101" => return "01010111" or shift_mask; -- W
-- ...
```

3.9.3 Klávesa Caps Lock

Jako další vylepšení se nabídlo implementovat klávesu Caps Lock. Její implementace je podobná implementaci klávesy Shift. Jediný rozdíl je ten, že příznak stisknutí klávesy Caps Lock se nevymaže po stisknutí následující klávesy. V procesu starajícím se o PS/2 vstup jsem si tedy vytvořil další proměnnou `caps_lock_button`, která drží aktuální stav Caps Locku. Při stisknutí klávesy Caps Lock se tato proměnná přepne. Do funkce `ScancodeToAscii` jsem začal odesílat jako příznak stisknuté klávesy Shift dvě proměnné. Proměnnou `shift_button` a `caps_lock_button` spojené logickou operací XOR. Výsledkem je tedy chování, že při stisknutí klávesy Caps Lock se píší velká písmena, při stisknutí klávesy Shift se následující písmeno napíše obráceně, než jak určuje stav Caps Locku. Tedy velmi podobně, jak je tomu u psaní na počítači.

Návrh chování klávesy Shift byl takový, že se nejdřív klávesa Shift stiskne a pustí, následně se stiskne klávesa písmena, které se vypíše jako velké písmeno (či malé, pokud je zapnutý Caps Lock). Při testování jsem zjistil, že je možné dokonce stisknout klávesu Shift a písmeno současně a výsledkem bude stále vypsání velkého písmena. Bohužel to ale už neplatí pro držení klávesy Shift a psaní více písmen po sobě, k tomu je vhodné použít klávesu Caps Lock. Jako poslední detail jsem ještě nastavil to, že na LED diodě 0 na vývojové desce bude zobrazen aktuální stav Caps Locku (podobně tomu, jak to funguje u běžných počítačů s indikační LED diodou přímo na klávesnici).

3.10 Výsledný stav projektu

K vývojové desce Spartan-3E je připojena PS/2 klávesnice. Po naprogramování desky se inicializuje a nakonfiguruje displej. Poté je možné klávesnicí psát na displej. Lze vypsat všechna malá a velká písmena (bez diakritiky), čísla a mezery. Fungují klávesy Shift a Caps Lock k přepínání velikosti písmen. Na LED diodách desky se ukazují částečné scankódy mačkaných kláves vyjma LD0 (nejvíce vpravo), na které je ukazován aktuální stav přepnutí funkce Caps Lock. Pomocí klávesy Enter je možné se přesunout na začátek řádku, ve kterém se aktuálně nenachází kurzor. Vývojová deska s na klávesnici napsanou ukázkovou zprávou je vidět na obrázku 3.6.

3.11 Možná rozšíření

V kódu by bylo možné přidat detekci dalších scankódů. Přidání speciálních znaků s jednobajtovým scankódem by bylo velmi jednoduché, stačilo by přidat odpovídající řádky do funkce pro překlad scankódů na ASCII kódy. V práci jsem implemenotval klávesu Shift (a Caps Lock) pouze pro psaní velkých a malých písmen. Jenoduchou úpravou by se daly tyto funkce rozšířit i pro již zmiňované speciální znaky.

Jiné vylepšení, které mě napadá, je mazání celého displeje pomocí jednoho

Obrázek 3.6: Výsledný stav projektu, vlastní fotografie

tlačítka na klávesnici. Implementace této funkce by mohla být velmi podobná mojí implementaci tlačítka Enter s tím rozdílem, že místo odesílání příkazu na změnu řádku by se displeji odeslal příkaz pro jeho vymazání.

Bylo by také možné zlepšit orientaci v textu např. pomocí šipek, jejichž implementace by byla již trochu složitější.

Kapitola 4

Závěr

Cílem práce bylo vytvořit kód v jazyce VHDL, který se bude starat o PS/2 klávesnicový vstup a následně vypíše na LCD displej znak stisknuté klávesy.

V práci byly popsány základní pojmy jako třeba FPGA nebo VHDL. Byl popsán protokol PS/2 včetně struktury přenášených dat a popisu scankódů. Pro komunikaci s displejem bylo potřeba popsat základní vlastnosti řadiče HD44780. Ke spojení kódu starajícího se o klávesnicový vstup s kódem starajícím se o LCD výstup bylo potřeba popsat teoretické možnosti propojení jednotlivých částí VHDL kódu.

Praktická část se zaměřuje na popis samotného postupu. Obsahuje popis zapisování dat na vestavěný LCD displej a popis čtení dat z PS/2 sběrnice včetně ukázek zdrojových kódů. Důležitou součástí praktické části jsou popisy problémů, které se vyskytly při práci. Praktická část popisuje konkrétní kroky vedoucí k úspěšnému řešení daných problémů a vysvětlení příčin problémů. Dále jsou zde popsány funkce a vylepšení kvality programu jako například implementace klávesy Shift či Enter. Na konci praktické části se nachází shrnutí výsledného stavu projektu a možnosti případného budoucího rozšíření.

V práci se podařilo napsat VHDL program, který zpracovává PS/2 vstup a vypisuje stisknutá písmena na LCD displej. Podařilo se implementovat několik dalších funkcí nad rámec zadání.

Seznam použitých zkratek

- ASCII (American Standard Code for Information Interchange) – Tabulka definující znaky anglické abecedy a jejich kódy
- ASIC (Application Specific Integrated Circuit) – Integrovaný obvod pro specifickou aplikaci
- FPGA (Field Programmable Gate Array) – Programovatelné hradlové pole
- HDL (Hardware Description Language) – Jazyk pro popis hardwaru
- LCD (Liquid Crystal Display) – Displej s tekutými krystaly
- LED (Light-Emitting Diode) – Elektroluminiscenční dioda
- LSB (Least Significant Bit) – Nejméně významný bit
- VHDL (Very High Speed Integrated Circuit Program Hardware Description Language) – Konkrétní představitel jazyka pro popis hardwaru

Literatura

[1] SCHELLE, Donald. *Serialize your HD44780 liquid crystal display.* Electronic Design. 2005, 2005(53), 47. ISSN 0013-4872.

[2] Xilinx, Inc. *Spartan-3E FPGA Starter Kit Board User Guide* [online]. In: . 20. 1. 2011, s. 43–54, 63–67 [cit. 2021-12-09]. Dostupné z: https://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf.

[3] TROCHIMIUK, Maciej. *FPGA programming—how it works and where it can be used* [online]. In: . 30. 4. 2021, [cit. 2022-05-18]. Dostupné z: https://codilime.com/blog/fpga-programming-how-it-works-and-where-it-can-be-used/.

[4] CHAPWESKE, Adam. *The PS/2 Mouse/Keyboard Protocol* [online]. In: . 9. 5. 2003, s. 3–6 [cit. 2021-02-03]. Dostupné z: https://www.avrfreaks.net/sites/default/files/PS2%20Keyboard.pdf

[5] MILLS, Matt. *PS / 2 Port on Motherboards: What it is for and Why it Matters* [online]. In: . 6. 8. 2020, [cit. 2021-02-08]. Dostupné z: https://itigic.com/ps-2-port-on-motherboards-why-it-matters/

[6] ASIX, s.r.o. *Referenční příručka k logickému analyzátoru SIGMA2* [online]. In: . 2. 7. 2021, [cit. 2022-04-21]. Dostupné z: https://asix.tech/_analyzers/sigma2_cz.pdf

[7] MERRICK, Russell. *Variables vs. Signals in VHDL* [online]. In: . 10. 1. 2016, [cit. 2022-05-10]. Dostupné z: https://www.nandland.com/vhdl/tips/variable-vs-signal.html

Seznam souborů na CD

Obsah přiloženého CD:

- Text bakalářské práce ve formátu PDF
- VHDL kód programu
- Projekt vytvořený v programu Xilinx ISE
- Stažené materiály ve formátu PDF
- Fotografie a obrázky